Project 2: Convolutional Neural Networks

Urszula Białończyk, Olaf Werner

April 2021

Contents

1 Introduction

Image recognition is one of the most famous examples of usage of machine learning. Well trained nets can recognize images with almost perfect accuracy. However ordinary neural networks are not perfect for the task because even very low resolution image is made of hundreds if not thousands pixels and each pixel is combination of 3 colors. That makes image recognition a problem of with thousands variables. To solve such problems special classes of neural networks has been designed: Convolutional Neural Networks and later Residual Networks. We created our networks to participate in Kaggle competition CIFAR-10 - Object Recognition in Images [1]. We tested many parameters for our neural networks mainly depth and width. We also tested many techniques to prevent overfitting.

2 Data set description

The CIFAR-10 data consists of 60,000 32×32 color images (Fig. 1) in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data.

To discourage certain forms of cheating 290,000 junk images in the test set were added. These images are ignored in the scoring in Kaggle [1].

The label classes in the dataset are:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

Figure 1: Example image from training set (airplane class)

3 Theoretical part

3.1 VGG

VGG [3] is a famous Convolutional Neural Network. One of our tested architectures were inspired by it. Main idea of this network is to use small convolutions layer in succession followed by 2×2 max pooling. Authors prove that such approach is better than using few layers with big filters. Using small filter size is especially relevant for our Kaggle challenge [1] where we have very low resolution images. Our main building block will be two convolution layers with 3×3 filters and same padding followed by 2×2 max pooling. We use batch normalization. First building block will have 32 filters and every next building block will have double number of filters than previous 2. Last part of the network will be composed of one fully connected layer with amount of neurons equal to number of filters in the previous convolutional layer and output layer made of 10 neurons, one for each class 3.

3.2 ResNet

Residual network, widely known as ResNet, was first introduced in 2015 in [**Resnet**]. It can be viewed as an extention of VGG nets since it follows their full 3×3 convolutional layer design. The core idea of ResNets is a residual block - every layer contains the identity function as one of its elements. The architecture of a regular block and a residual block can be seen at the below picture.
What is worth noting at the very beginning is that such modification of a regular block doesn't require to find any additional parameters. Although it seems as a very simple idea, it turns out that a net consisting of such blocks is a solution to a problem called *degradation* - when the network's depth increases, accuracy gets saturated and then degrades rapidly. An example of such behaviour is presented below.

This is especially a problem with image recognition tasks since deeper networks tend to perform better. They can learn more details than the shallow ones and this helps during training. With the degradation problem in mind, the authors of ResNet wanted to build a network that could be really deep with low train and test errors. They introduced a deep residual learning framework in which each

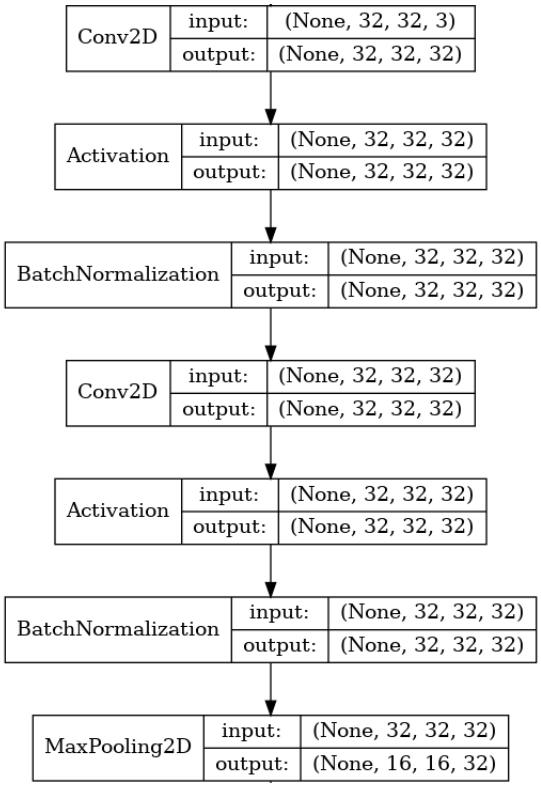

Figure 2: Basic building block of our network.

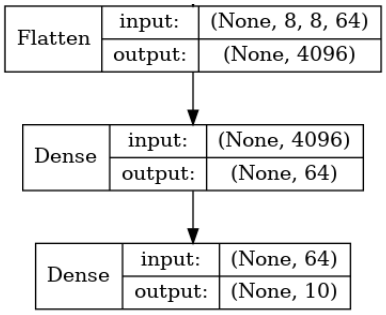

Figure 3: Final layers of our architecture.

few stacked layers fit a residual mapping $H(x)$. The stacked nonlinear layers fit another function, F(x) := H(x) - x, which means that the original mapping is transformed into $F(x) + x$. Such formulation can be realized by feedforward neural networks using so called "shortcut connections" i.e. connections skipping at least one layer. In the case of ResNets, the shortcut connections perform

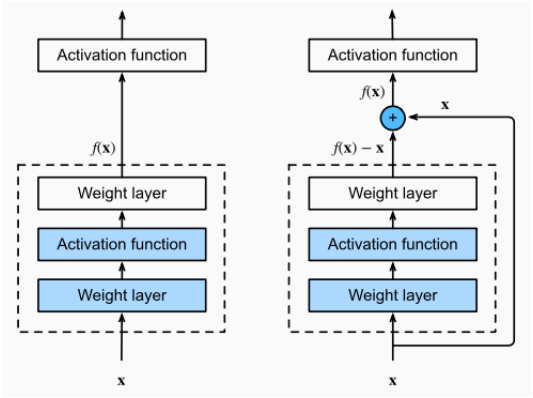

Figure 4: A traditional block (left) and a residual block (right).

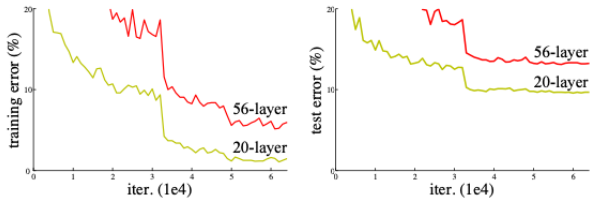

Figure 5: Degradation of train (left) and test (right) errors.

identity mapping, and their outputs are added to the outputs of the stacked layers (Fig. 1). What is important is that the entire network can be trained end-to-end by SGD with backpropagation, so it doesn't require any more tools than building e.g. a VGG net. Moreover, it turned out that it is easier to optimize the residual mapping than to optimize the original, unreferenced one.

4 Experiments

4.1 VGG

Some of the training settings will be the same for all of our networks.

- Activation function: ReLU
- Optimizer: Adam with learning rate = 0.001, exponential decay rate for the 1st moment = 0.9, exponential decay rate for the 2st moment = 0.999 (Default Keras settings)
- Loss function: categorical crossentropy
- Batch size: 64

- Epochs: 100
- Validation dataset size: 10000 (20% of available data)

4.1.1 Most basic network

We start with our most basic network (Fig. 6) made with just one bulding block 2. We immediately see very serious problem. On training set our model achieves near perfect results while on validation it achieves around 60%. Such phenomenon is called "overfitting". Before we will increase the size of our network we must first take care of overfitting as overfitting gets worse as number of trainable parameters of model increases [4].

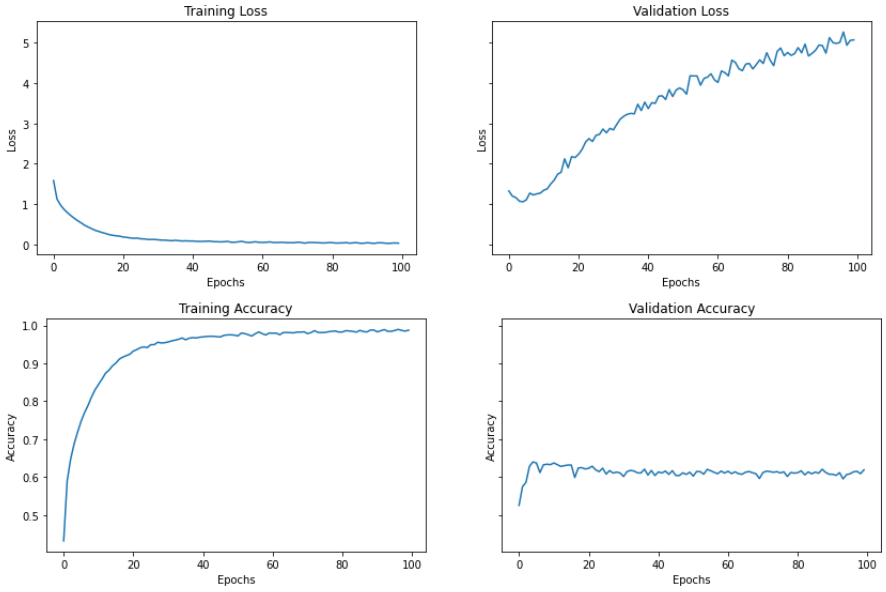

Figure 6: Training and Validation losses and accuracies for CNN with one block and no regularization

4.1.2 Most basic network with data augumentation

One of the solutions to the overfitting is using data augmentation. Data augmentation is modifying original data with slight alternations to expand dataset. Thanks to this we can prevent overfitting and improve accuracy.

We apply randomly following modifications with each batch

- horizontal flip (mirror image)

- move image up to 3 pixels up or down
- move image up to 3 pixels left or right
- brighten or darken image up to 10% of original brightness
- rotate image up to 15 degrees

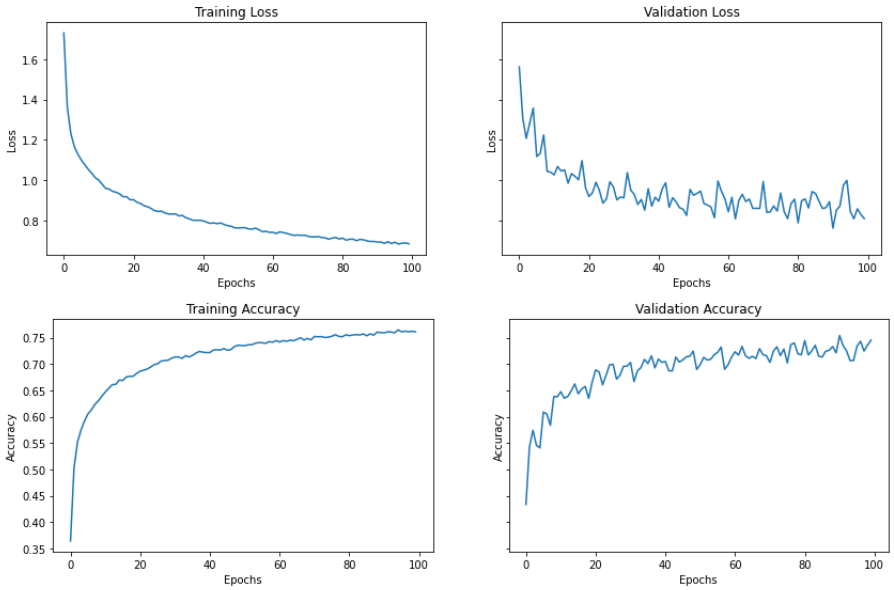

Figure 7: Training and Validation losses and accuracies for CNN with one block and data augmentation

As we can see we improved accuracy to 75% and training and validation losses are more similiar. From now on we will use data augmentation for all next models.

4.1.3 Network with two building blocks

Network achieves 80% accuracy on validation dataset which is better than previous model, however we again see gap between training and validation losses. Therefore we must include another technique to prevent overfitting.

4.1.4 Network with two building blocks and regularization

Another way of preventing overfitting is called regularization. Regularization is adding additional penalty to cost function. This penalty is for size of weights. Thanks to this weights are smaller and model is more generalizing. We use

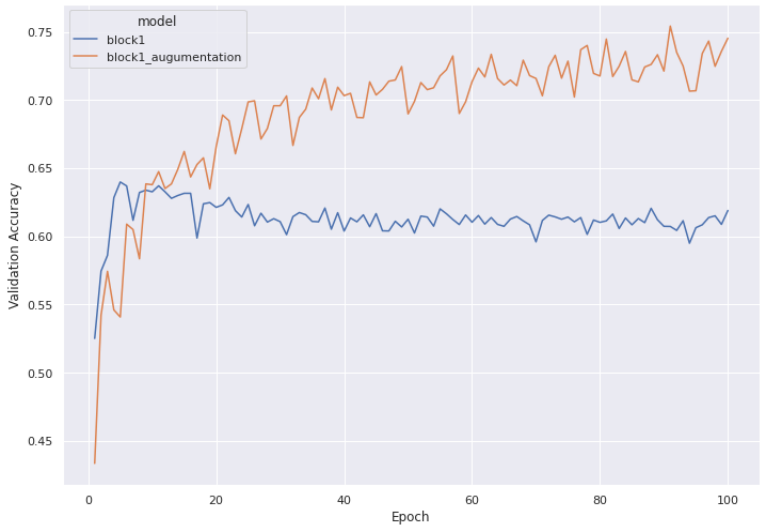

Figure 8: Comparison of Validation accuracies of CNNs with one block with and without data augmentation

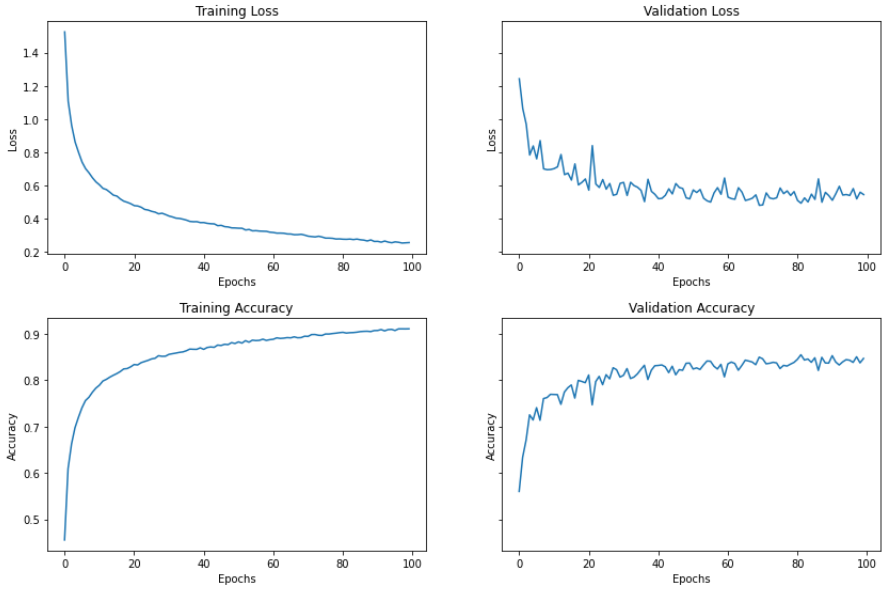

Figure 9: Training and Validation losses and accuracies for CNN with two blocks and data augmentation

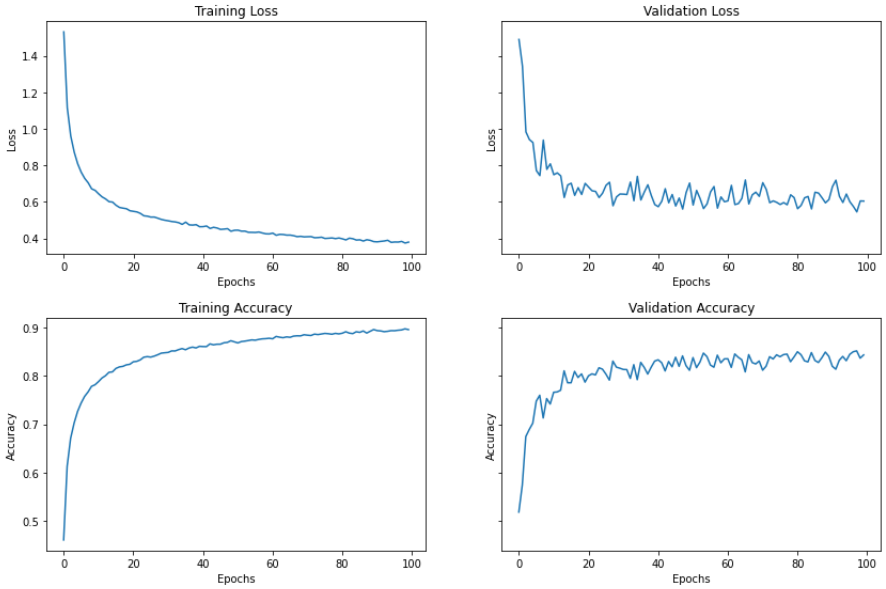

Figure 10: Training and Validation losses and accuracies for CNN with two blocks, data augmentation and L2 regularization

L2 penalty. That means we add to cost function this term: $\lambda \sum \beta^2$ where λ is magnitude of penalty and β are weights of connections. We used $\lambda = 0.001$.

There were minor improvements in gap between validation and training datasets, also accuracy of this network was similiar to previous network. However we still must do more to prevent overfitting.

4.1.5 Network with two building block and dropout

Dropout is technique to prevent overfitting by randomly shutting down connections during training. When network calculate results some of the neurons and their connections are randomly becoming inactive. Thanks to this network needs to learn how to use different set of neurons and cheating by simply encoding training dataset becomes harder.

We can see major improvement in reducing gap between validation and training datasets however there is significant drop in accuracy on validation dataset. This is expected as fighting overfitting can be at the cost of accuracy. We will keep adding block to improve accuracy.

4.1.6 Network with three building blocks

We see an significant improvement in accuracy on validation dataset. This model is better than previous ones. Also gap between loss on validation and

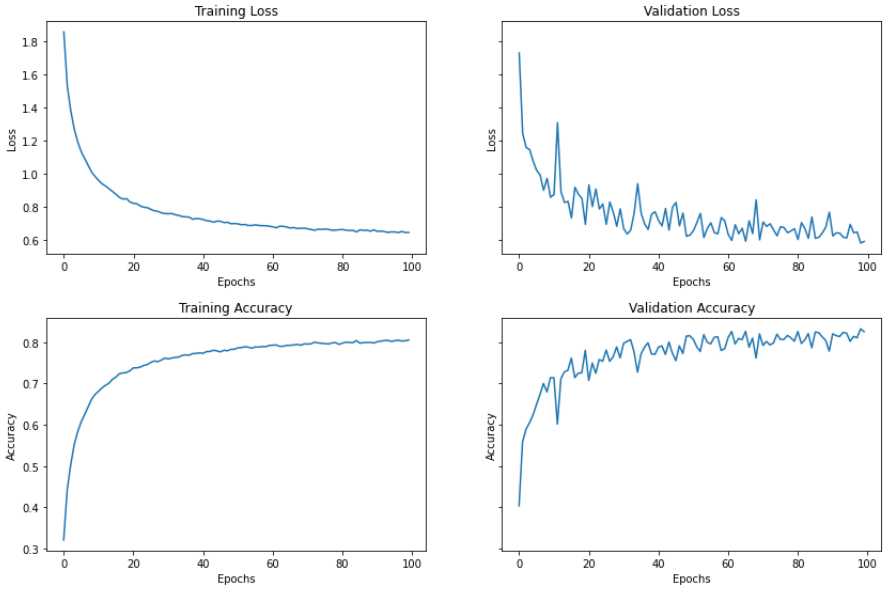

Figure 11: Training and Validation losses and accuracies for CNN with two blocks, data augmentation, L2 regularization and dropout

Figure 12: Comparison of Validation accuracies of CNNs with two blocks with different overfitting prevention methods

Figure 13: Comparison of Differences between Validation and Training Losses of CNNs with two blocks with different overfitting prevention methods

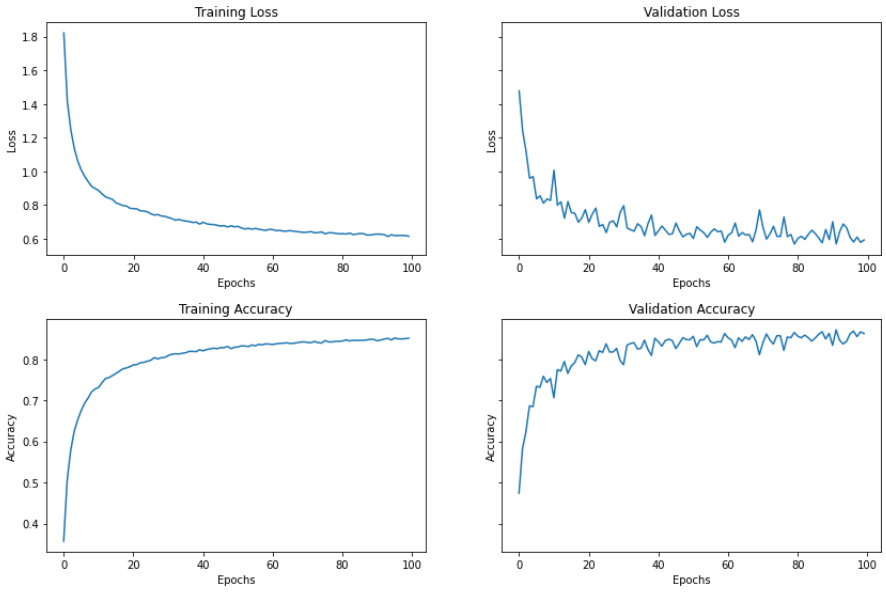

Figure 14: Training and Validation losses and accuracies for CNN with three blocks, data augmentation, L2 regularization and dropout

training is small.

4.1.7 Network with four building blocks

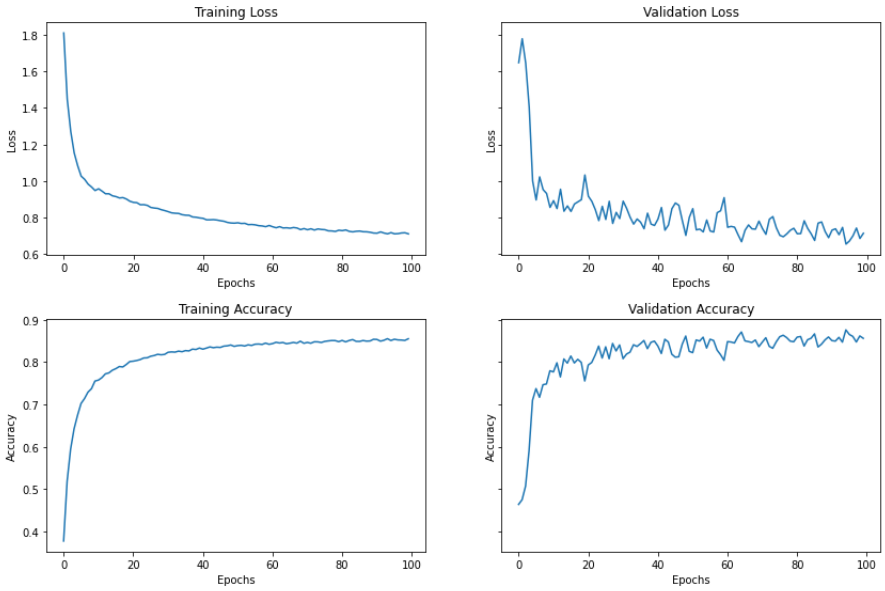

Figure 15: Training and Validation losses and accuracies for CNN with four blocks, data augmentation, L2 regularization and dropout

Results are very similar and it is impossible to say which model is better, because accuracy on validation dataset is jagged. but technically this model achieved highest score ever in 94 epoch.

4.1.8 Network with five building blocks

Results are also very similar to previous two networks, however they are slightly worse.

4.1.9 VGG summary

In the end the best network turned out to be an block 4 network. It achieved 85.16% on competition [1]. We learned many things during creating network. First of all image augmentation is a great way to improve accuracy and prevent overfitting. Other overfitting prevention techniques also proved successful. We also checked that adding more layers to the network improves accuracy until it starts giving diminishing returns and later even worse results.

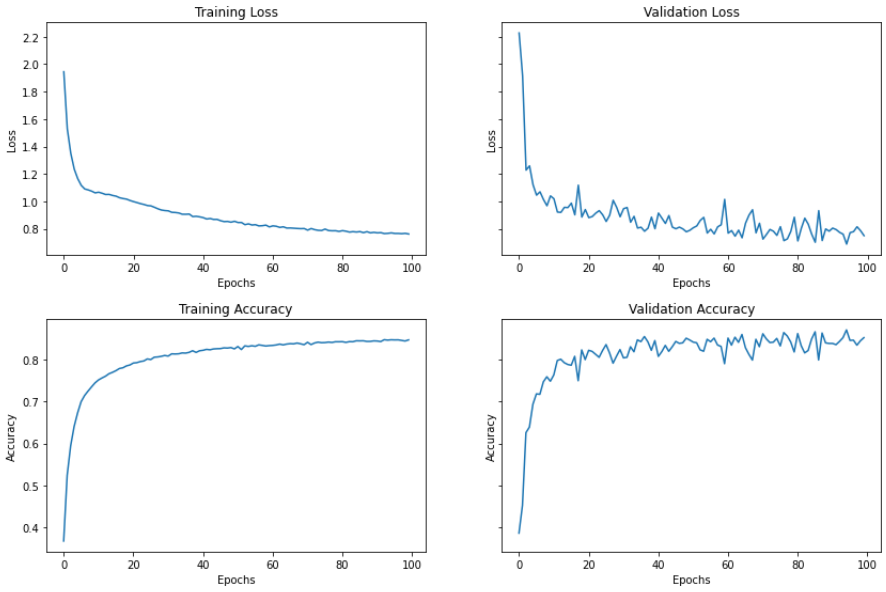

Figure 16: Training and Validation losses and accuracies for CNN with five blocks, data augmentation, L2 regularization and dropout

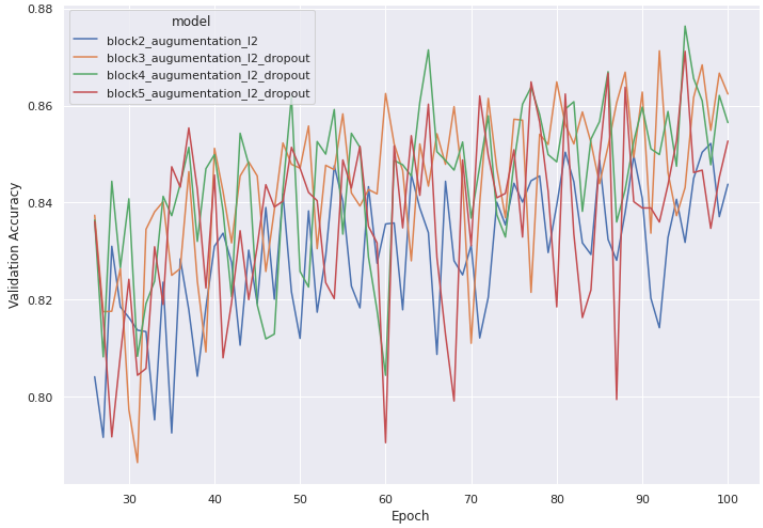

Figure 17: Comparison of Validation accuracies of CNNs with different number of blocks

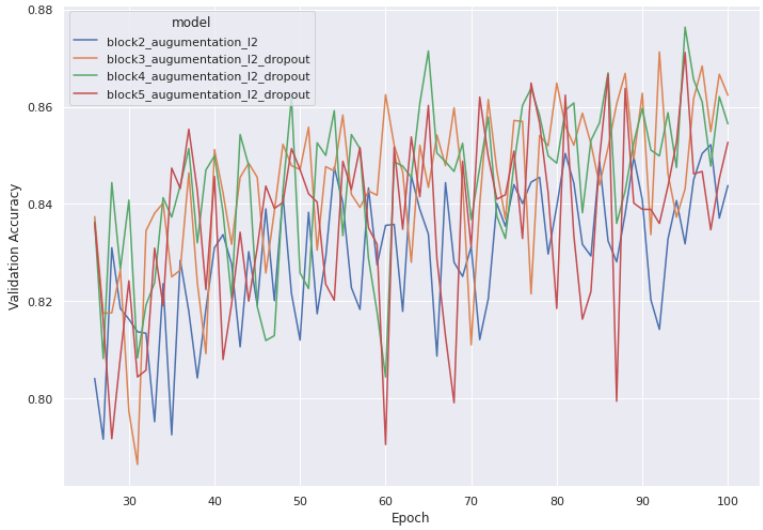

Figure 18: 4 blocks VGG results on Kaggle

4.2 ResNet

Since experiments with VGG nets took a lot of time, we decided to look for solutions that could give equally good or even better results in shorter training time. We were new to the topic, so we followed two tutorials. The first one can be found on Keras Web page while the second one, which we can highly recommend, can be found in [2].

4.2.1 Basic ResNet

The first ResNet was a simple one, built in Keras from scratch. We wanted to understand what was going on in the network's architecture and Keras seems more user-friendly to do so. When it comes to data preprocessing, we used a 'standard' approach: first we performed the normalization, then we set width and height shift ranges to 0.1 and allowed random horizontal flip.
When it comes to the network's architecture, we tested the 'original' so we stacked 2 x (3 x 3) Conv2D-BN-ReLU. At the beginning of each stage, the feature map size was halved by a convolutional layer with strides=2 and the number of filters was doubled. During each stage, the layers had the same number of filters.
The other network's features worth mentioning are:

- Convolutional layer: we used 'he_normal' kernel initializer and l2 kernel regularizer.
- Residual layer: we used relu activation function, batch normalization and set the parameter 'conv_first' to True which means that we used conv-bn

activation, not the other way round.

- For training, we used categorical crossentropy loss function and Adam optimizer.

We trained the network for 200 epochs. The best accuracy on the validation dataset was 0.91220 and it was achieved in the epoch 91. Since then the accuracy on the validation set oscilated around 0.91 while the accuracy on the training set was constantly improving (in the last few epochs it was around 0.985), we concluded that it was a classical example of overfit. Nevertheless, 0.91 is a pretty good result with almost no data augmentation and little reguralisation techniques so it is obvious that ResNets are a very powerful tool.
We used a checkpoint which saved the best model in terms of accuracy on the validation set but unfortunately we lost the Internet connection before downloading it to the local machine and all of the created resources were lost. Since the training of 1 epoch lasted around 40 seconds and we didn't plan to use this model for prediction on the original dataset, we decided not to train the model again. Therefore we didn't plot the results but a printed data frame can be found in the notebook named 'ResNet.ipynb'.

4.2.2 Fast ResNet

As mentioned earlier, to conduct the experiments from this section, we mostly based on the tutorial from [2]. The whole idea of this series of blog posts was to build an effective (with accuracy over 94 % on the validation set) but fast ResNet. Throughout the series the authors give small tips on how to search for the optimum network structure which can be very useful in the future projects. The main difficulty that we encountered was that the code was on a very high level and it was wrote using PyTorch which is less intuitive than Keras for the beginners. Therefore the first step here was the code analysis so as to be able to modify it accordingly to our task.
The baseline model consisted of three layers: residual - convolutional - residual. The convolutional blocks use relu activaion function and, as in the previous section, we used conv-bn activation. When it comes to some data augmentation, we used a whitening filter and cropping. For such configuration, we obtained the following results:

As it can be seen from the plots, the model overfits to the training data. We This means that it might be worth to apply some regularization.

4.2.3 Fast ResNet - data augmentation experiments

We decided to manipulate with data augmentation. At first we added horizontal flipping since it seems to be a usual practice, then we added cutout (which means that parts of the input will be 'dark'). We learned that for cifar10 cutout proved to be quite effective, more than the usual technique, dropout. We decided to use different parameters: 4x4, 8x8 and 16x16. The results are presented

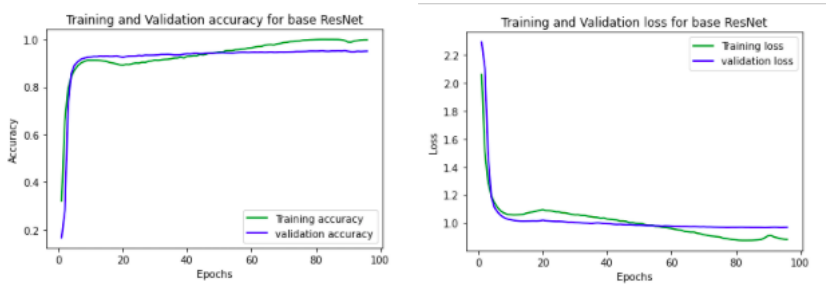

Figure 19: Accuracy and loss for a basic, 3-layer model.

below.

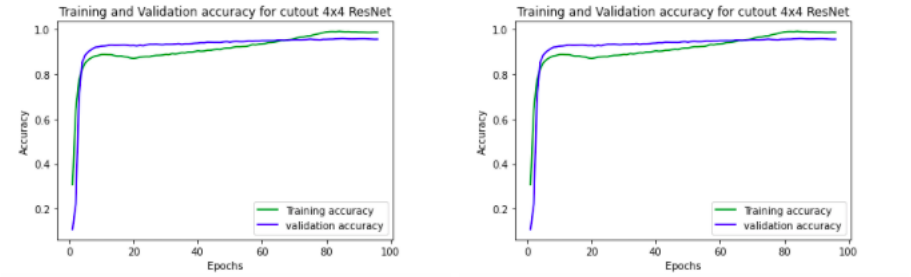

Figure 20: Accuracy and loss for a 3-layer model with 4x4 cutout.

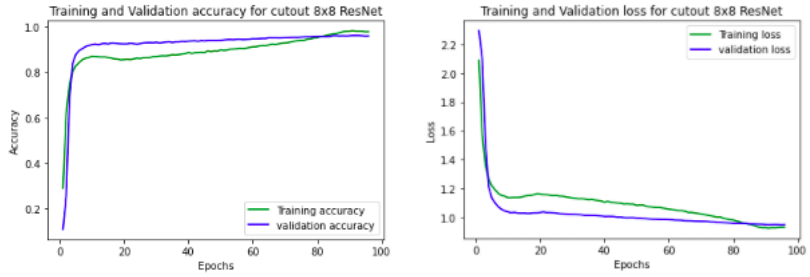

Figure 21: Accuracy and loss for a 3-layer model with 8x8 cutout.

It can be noticed that the best results were obtained for cutout 16x16 but all of the parameters prevent overfitting. It's also worth noting that the accuracy

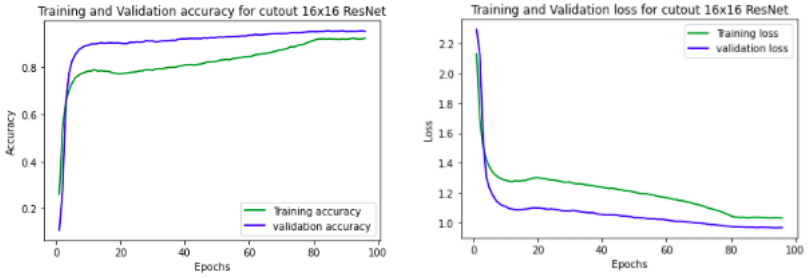

Figure 22: Accuracy and loss for a 3-layer model with 16x16 cutout.

tends to be higher on the validation set than on the test set. It might be because we made the task more difficult for the classifier by using cutout while the images in the test set are quite easy for a classifier trained in such a way.

4.2.4 Fast ResNet - architecture experiments

We tried to expand the network. Apart from a 3-layer ResNet, we also trained 4 and 5 layer ones. We didn't use cutout here so that we could see whether enlarging the net without additional regularization helps with the training. The results can be seen on the below plots.

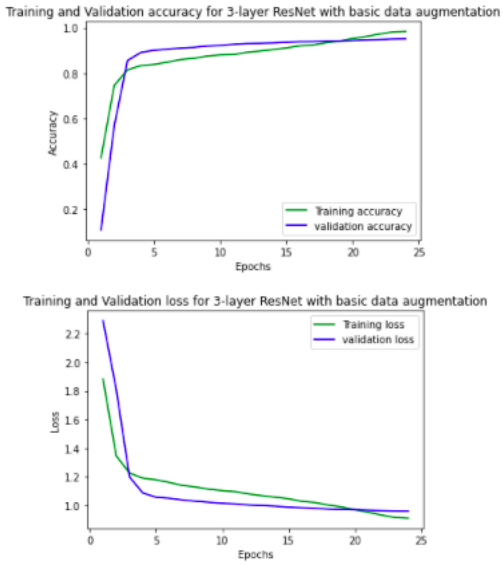

Figure 23: Accuracy and loss for a 3-layer model.

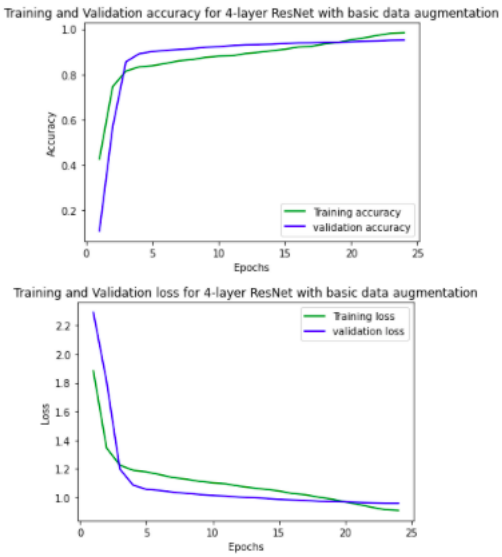

Figure 24: Accuracy and loss for a 4-layer model.

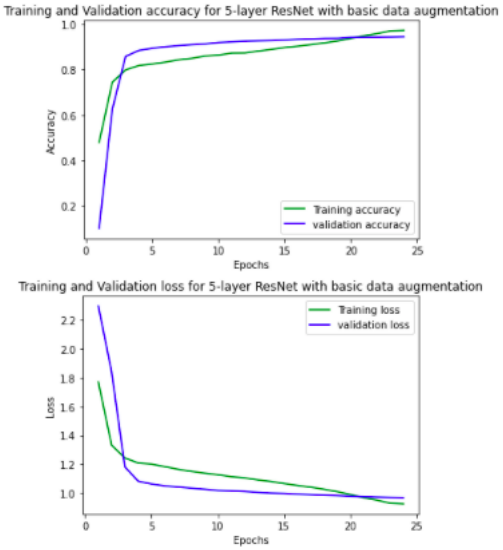

Figure 25: Accuracy and loss for a 5-layer model.

We can see that there isn't a big difference between the accuracies scored on the validation set. However, the training times are different.For the shortest, 3-layer architecture, the training of 1 epoch lasts $\tilde{1}0$ sec. For the 5-layer one, it is over twice as high.

4.2.5 Final models

We submitted 3-layer Network with 8×8 cotout. This model achieved 94% accuracy on Kaggle.

Figure 26: ResNet result on Kaggle.

5 Summary

We managed to achieve pretty good models. Resnets proved to be better than VGG but this is not surprising considering that ResNets are more advanced. We showed that preventing overfitting is importent and ways it can be done especially data augmentation.

References

[1] *CIFAR-10 - Object Recognition in Images*. `https://www.kaggle.com/c/cifar-10/`.

[2] *How to train your ResNet*. `https://myrtle.ai/learn/how-to-train-your-resnet/`.

[3] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: `1409.1556 [cs.CV]`.

[4] Xue Ying. "An Overview of Overfitting and its Solutions". In: *Journal of Physics: Conference Series* 1168 (Feb. 2019), p. 022022. DOI: `10.1088/1742-6596/1168/2/022022`.